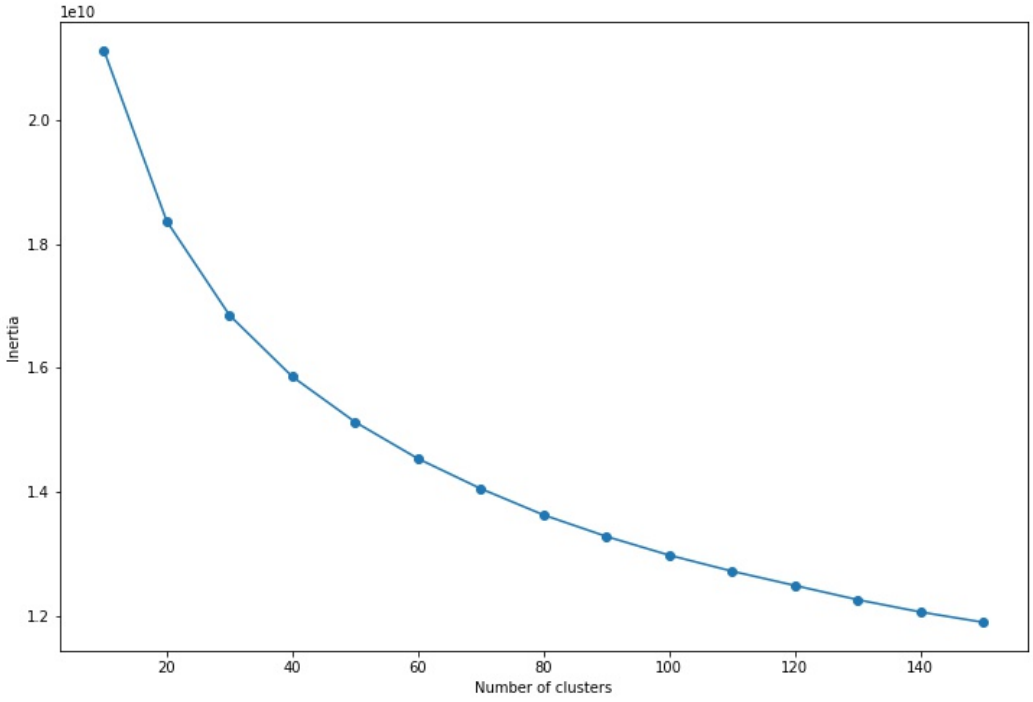
Create a Bag of words based matching

Dataset (train and test): https://www.kaggle.com/zalando-research/fashionmnist/data

Clustering Algorithm: KMeans++

Selecting optimal value of number of clusters (k=105)

I ran the algorithm for the number of clusters value 10,20,...150 and plotted the inertia vs number of cluster graphs.



From above plot, till **100 to 110** number of clusters, the decrease in inertia was considerably changing, **but hitting plateau later.** As a higher number of clusters is computationally expensive, so I choose any value between 100 to 110 (**105**) for better performance and result.

Descriptor used

I have used the SIFT keypoint descriptor. Since it is removed from most of the libraries due to license expiration, I am still able to get it only in opencv. (Opencv I used because the depreciated SIFT keypoints detector is still working. No other function of opencv is used in the assignment).

In addition, I tried **OBR, DAISY** etc keypoints feature extractor of skimage, but it failed to give any keypoints as the image is 28X28 is off very low resolution.

MatchHistogram()

Chi-Square distance is used to find the distance between the two input histograms. Chi square was giving better accuracy (~39%) compared to euclidean distance which was giving ~25% accuracy.

Results:-

Average accuracy: %38.60434640165993

Class based performance:

0 Accuracy= % 74.0701381509033 Precision= 0.2236842105263158 Recall: 0.740701381509033
1 Accuracy= % 41.97324414715719 Precision= 0.3653566229985444 Recall: 0.4197324414715719
2 Accuracy= % 47.794117647058826 Precision= 0.33211678832116787 Recall: 0.47794117647058826
3 Accuracy= % 22.308546059933406 Precision= 0.32682926829268294 Recall: 0.22308546059933407
4 Accuracy= % 19.27835051546392 Precision= 0.3702970297029703 Recall: 0.19278350515463918
5 Accuracy= % 46.790890269151134 Precision= 0.6034712950600801 Recall: 0.46790890269151136
6 Accuracy= % 8.360477741585234 Precision= 0.23692307692307693 Recall: 0.08360477741585233
7 Accuracy= % 40.08438818565401 Precision= 0.5337078651685393 Recall: 0.4008438818565401
8 Accuracy= % 35.66289825282631 Precision= 0.7229166666666667 Recall: 0.3566289825282631
9 Accuracy= % 49.442755825734544 Precision= 0.8160535117056856 Recall: 0.49442755825734547

```
]: result_matrix = KNN(histogram_bovw_train,histogram_bovw_test)
```

```
]: def performance_matrix(result_matrix):
    avg_acc = (result_matrix[1] / result_matrix[0]) * 100
    print("Average accuracy: %" + str(avg_acc))
    print("\nClass based performance: \n")
    for key,value in result_matrix[2].items():
        accuracy = (value[0] / value[1]) * 100
        precision=(value[0]/(value[0]+value[2]))
        recall=(value[0]/(value[0]+value[3]))
        print(key,"  Accuracy= %",str(accuracy), "Precision=",str(precision), "Recall:",str(recall))
```

```
]: performance_matrix(result_matrix)
Average accuracy: %38.60434640165993

Class based performance:

0   Accuracy= % 74.0701381509033 Precision= 0.2236842105263158 Recall: 0.740701381509033
1   Accuracy= % 41.97324414715719 Precision= 0.3653566229985444 Recall: 0.4197324414715719
2   Accuracy= % 47.794117647058826 Precision= 0.33211678832116787 Recall: 0.47794117647058826
3   Accuracy= % 22.308546059933406 Precision= 0.32682926829268294 Recall: 0.22308546059933407
4   Accuracy= % 19.27835051546392 Precision= 0.3702970297029703 Recall: 0.19278350515463918
5   Accuracy= % 46.790890269151134 Precision= 0.6034712950600801 Recall: 0.46790890269151136
6   Accuracy= % 8.360477741585234 Precision= 0.23692307692307693 Recall: 0.08360477741585233
7   Accuracy= % 40.08438818565401 Precision= 0.5337078651685393 Recall: 0.4008438818565401
8   Accuracy= % 35.66289825282631 Precision= 0.7229166666666667 Recall: 0.3566289825282631
9   Accuracy= % 49.442755825734544 Precision= 0.8160535117056856 Recall: 0.49442755825734547
```

```
]:
```

Fig: Screenshot of Notebook

Why accuracy is low:-

The reason for the low performance (accuracy) in some of the **classes (say class 6)** is because of the feature extractor. **SIFT feature extractor failed to give any keypoints** (or extremely low (1 to 3) key points) for so small images (28X28 size). Thus, it leads to misclassification and affects other accuracy.

The same algorithm can give better accuracy if tested on a good resolution image dataset.

Files added:-

- **One.py**
- **RunALL.py**
- **Report.ipynb** You can directly see the results of the same code I run on python notebook.
- **clusters_center_points.npy:** After running the program, these are the center points I got. (Array of 105 that is 105 cluster)

- **closest_visual_word_dictionary_with_class.npy :** It contains the closest visual words dictionary with the class (cluster).

How you can test algorithm quickly:-

- Either please reduce the data size, **or please go to one.py, then uncomment line number 36,37.** However, It will compromise the accuracy then.

Functions Details:-

- *CreateDictionary()* – It computes and saves the visual dictionary.
- *ComputeHistogram()* – It takes as input a feature vector and visual dictionary matrix and then generates the histogram using soft assignment (giving the weight to the next nearest neighbor)
- *MatchHistogram()* – the function compares two histograms and then returns the distance.
- RunAll.py, which extracts features from the images and then calls the *CreateDictionary()* function. The closest (visual) word to the mean of the cluster is saved in a folder directory. This is for visualising what do the dictionary words represent? Then the script calls the *ComputeHistogram()* function to create the histograms for all the training and Test images. *MatchHistogram()* then is called for the Test set images and the Label (category) for each Test image is generated by assigning the class of the nearest neighbor (in the Training set)
- O*verall classification accuracy, class wise accuracy, precision and recall* is also displayed.